A type system for PSPACE derived from light linear logic

Lucien Capdevielle *
ENS de Lyon

We present a polymorphic type system for lambda calculus ensuring that well-typed programs can be executed in polynomial space: *dual light affine logic with booleans* ($DLAL_B$). To build $DLAL_B$ we start from *DLAL* (which has a simple type language with a linear and an intuitionistic type arrow, as well as one modality) which characterizes FPTIME functions. In order to extend its expressiveness we add two boolean constants and a conditional constructor in the same way as with the system STA_B in [7].

We show that the value of a well-typed term can be computed by an alternating machine in polynomial time, thus such a term represents a program of PSPACE (given that PSPACE = APTIME ([5])).

We also prove that all polynomial space decision functions can be represented in $DLAL_B$.

Therefore $DLAL_B$ characterizes PSPACE predicates.

1 Introduction

The topic of this paper is Implicit Computational Complexity which is the field of study of calculi and languages with intrisic complexity-theoretical properties. One of the main issues of this field is to design programing languages with bounded computational complexity. Historically, there have been various approaches:

- restriction of recursive schemes ([4], [12])
- interpretation methods for first order interpretational languages ([13], [9])
- variations of linear logic and proofs-as-programs Curry-Howard correspondence ([8], [1] and [11])

The latest approach has led to the design of type systems for λ-calculus such that the set of all well-typed terms corresponds to the class PTIME.

In this paper, we will present a type assignment system which guarantees that a program of the language is PSPACE and that all predicates of PSPACE can be encoded in this language.

Coming back to the approach of linear logic, it is based on the observation that the duplication rule is controlled by the logical connective "!". Moreover, the power of duplication is responsible for the complexity of normalization. Thus, by replacing the "!" with a weaker connective, one obtains systems with controlled duplications, and where normalization offers a complexity bound. Light Linear Logic (LLL, [8]) and Soft Linear Logic (SLL, [11]) are two examples of such systems.

First, the system *DLAL* ([2] and [3]) has been derived from LLL and then the system *STA* ([6]) from SLL. These systems are both characterizing PTIME. Then, in order to characterize PSPACE predicates, Gaboardi and al. have designed the system STA_B ([7]) by adding two boolean constants and a conditional constructor to the system *STA*. The goal of this paper is to see if it is possible to adapt this method in order to obtain a system characterizing PSPACE by modifying the system *DLAL*.

It is straightforward to define $DLAL_B$ starting from *DLAL* in an analogous way of STA_B is defined from *STA*. However, proving that the complexity bound of this system is polynomial is not obvious. In

*Partially supported by the project ANR-08-BLANC-0211-01 "COMPLICE".

Jean-Yves Marion (Ed.): Second Workshop on
Developments in Implicit Computational Complexity (DICE 2011)
EPTCS 75, 2012, pp. 33–46, doi:10.4204/EPTCS.75.4

fact, one difficulty is that the complexity bound of LLL and *DLAL* is proved by using a specific reduction strategy (level-by-level strategy) which is not compatible with the conditional we add to the language. Thus we will introduce an abstract alternating machine and a measure on the terms in order to prove the PSPACE bound. Thus we use the fact that PSPACE = APTIME ([5]) both in the completeness and the soundness parts of the proof (contrary to the proof that STA_B characterizes the predicates of PSPACE where PSPACE = APTIME is only used for the completeness).

The paper is organized as follows. We first give the definition of the system $DLAL_B$ and some properties of this system in Section 2. Then in Section 3 we give the proof that any well-typed term represents a predicate of APTIME. Finally, in Section 4 we prove that any predicate of APTIME is represented by a well-typed term.

2 λ-calculus with booleans and type assignment

In this section, we will first define $DLAL_B$, then we will give some classical properties which are true for terms well-typed in $DLAL_B$.

2.1 Definition of Λ_B and $DLAL_B$

We start from the λ-calculus of *DLAL* and will extend it with booleans and a conditional constructor in order to obtain $DLAL_B$ (analogous to [7]).

The language $\mathscr{L}_{DLAL_B}$ of $DLAL_B$ types is given by:

$$A,B ::= \alpha \mid A \multimap B \mid A \Rightarrow B \mid \S A \mid \forall \alpha.A \mid Bool.$$

$DLAL_B$ can be seen as a refinement of System F ensuring some complexity properties.
The language Λ_B of λ-terms with booleans is given by:

$$t,u,v ::= x \mid F \mid T \mid \lambda x.t \mid t\ u \mid if\ t\ then\ u\ else\ v.$$

The terms of Λ_B admit another type of reduction than the β-reduction, the δ-reduction which is the contextual closure of:

$$(if\ T\ then\ u\ else\ v) \xrightarrow{\delta} u$$
and
$$(if\ F\ then\ u\ else\ v) \xrightarrow{\delta} v.$$

Definition 1 *A term t of Λ_B can be written in a unique way as $M = N_0\ N_1\ ...\ N_m$ with $m \in \mathbb{N}$ and ($N_0 = x$ or $N_0 = \lambda x.t$ or $N_0 = if\ M_0\ then\ M_1\ else\ M_2$).*
The terms N_i are called elements of the canonical composition.

In order to prove the complexity bound, we have to adapt the classical notion of number of occurences in such a way that it is compatible with the additive rule (B e) of $DLAL_B$ (defined in Figure 1).

Definition 2 *The number of occurences of a variable in a term is inductively defined on the structure of the terms as follows: $no(x,x) = 1$, $no(x,y) = 0$, $no(x,F) = 0$, $no(x,T) = 0$, $no(x,\lambda y.t) = no(x,t)$, $no(x,\lambda x.t) = no(x,t)$, $no(x,t\ u) = no(x,t) + no(x,u)$, $no(x,if\ t_0\ then\ t_1\ else\ t_2) = \max_i no(x,t_i)$.*

$$\frac{}{;x:A\vdash x:A}\ (\mathrm{Id})$$

$$\frac{\Gamma;\Delta,x:A\vdash t:B}{\Gamma;\Delta\vdash \lambda x.t:A\multimap B}\ (\multimap \mathrm{i}) \qquad \frac{\Gamma_1;\Delta_1\vdash t:A\multimap B \quad \Gamma_2;\Delta_2\vdash u:A}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2\vdash t\ u:B}\ (\multimap \mathrm{e})$$

$$\frac{\Gamma,x:A;\Delta\vdash t:B}{\Gamma;\Delta\vdash \lambda x.t:A\Rightarrow B}\ (\Rightarrow \mathrm{i}) \qquad \frac{\Gamma;\Delta\vdash t:A\Rightarrow B \quad ;z:C\vdash u:A}{\Gamma,z:C;\Delta\vdash t\ u:B}\ (\Rightarrow \mathrm{e})$$

$$\frac{\Gamma_1;\Delta_1\vdash t:A}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2\vdash t:A}\ (\mathrm{Weak}) \qquad \frac{x_1:A,x_2:A,\Gamma;\Delta\vdash t:B}{x:A,\Gamma;\Delta\vdash t[x/x_1,x/x_2]:B}\ (\mathrm{Cntr})$$

$$\frac{;\Gamma,\Delta\vdash t:A}{\Gamma;\S\Delta\vdash t:\S A}\ (\S \mathrm{i}) \qquad \frac{\Gamma_1;\Delta_1\vdash u:\S A \quad \Gamma_2;x:\S A,\Delta_2\vdash t:B}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2\vdash t[u/x]:B}\ (\S \mathrm{e})$$

$$\frac{\Gamma;\Delta\vdash t:A}{\Gamma;\Delta\vdash t:\forall\alpha.A}\ (\forall \mathrm{i})\ (*) \qquad \frac{\Gamma;\Delta\vdash t:\forall\alpha.A}{\Gamma;\Delta\vdash t:A[B/\alpha]}\ (\forall \mathrm{e})$$

$$\frac{}{;\vdash F:Bool}\ (B_0\ \mathrm{i}) \qquad \frac{}{;\vdash T:Bool}\ (B_1\ \mathrm{i})$$

$$\frac{\Gamma;\Delta\vdash M_0:\S^n Bool \quad \Gamma;\Delta\vdash M_1:A \quad \Gamma;\Delta\vdash M_2:A \quad n\in\mathbb{N}}{\Gamma;\Delta\vdash if\ M_0\ then\ M_1\ else\ M_2:A}\ (B\ \mathrm{e})$$

Figure 1: Natural deduction system for $DLAL_B$

Examples: $no(x,(if\ x\ then\ x\ else\ x\ y)\ y)=1$
$no(y,(if\ x\ then\ x\ else\ x\ y)\ y)=2$.

For $DLAL_B$ typing we will handle judgements of the form $\Gamma;\Delta\vdash t:A$ (and $\Gamma\vdash_F t:A$ for System F). The intended meaning is that variables in Δ are (affine) linear, that is to say that they have at most one occurrence in the term, while variables in Γ are non-linear. We give the typing rules as a natural deduction system: see Figure 1 (the rules of $DLAL_B$ are those of $DLAL$ plus $(B_0\ \mathrm{i})$, $(B_1\ \mathrm{i})$ and $(B\ \mathrm{e})$).

We have:

- for $(\forall\ \mathrm{i})$: $(*)$ α does not appear free in Γ,Δ.
- in the $(\Rightarrow\ \mathrm{e})$ rule the r.h.s. premise can also be of the form $;\vdash u:A$ (u has no free variable).

Definition 3 *The depth of a $DLAL_B$ derivation $\mathscr{D}$ is the maximal number of premises of $(\S i)$ and r.h.s. premises of $(\Rightarrow e)$ in a branch of $\mathscr{D}$.*

Definition 4 *The l.h.s. premises of $(\multimap e)$, $(\Rightarrow e)$ and $(\S e)$ as well as the unique premise of $(\forall e)$ are called major premises. A $DLAL_B$derivation is $\forall\S$-normal if:*

- *no conclusion of a $(\forall i)$ rule is the premise of a $(\forall e)$ rule;*
- *no conclusion of a $(\S i)$ rule is the major premise of a $(\S e)$ rule;*
- *no conclusion of $(Weak)$, $(Cntr)$ and $(\S e)$ is the major premise of elimination rules: $(\multimap e)$, $(\Rightarrow e)$, $(\S e)$ and $(\forall e)$.*

Definition 5 *Let $\bar{\delta}$-reduction be the reduction defined by:*
Let t_0 be a closed term.
Let C be a context.

$C[if\ t_0\ then\ t_1\ else\ t_2] \xrightarrow{\bar{\delta}} t_0$
$C[if\ t_0\ then\ t_1\ else\ t_2] \xrightarrow{\bar{\delta}} C[t_1]$
$C[if\ t_0\ then\ t_1\ else\ t_2] \xrightarrow{\bar{\delta}} C[t_2]$

Examples:
$(\lambda x.(if\ (\lambda z.z\ F)\ then\ (x\ u)\ else\ y)\ v) \xrightarrow{\bar{\delta}} (\lambda z.z\ F)$
$(\lambda x.(if\ (\lambda z.z\ F)\ then\ (x\ u)\ else\ y)\ v) \xrightarrow{\bar{\delta}} ((\lambda x.(x\ u))\ v)$
$(\lambda x.(if\ (\lambda z.z\ F)\ then\ (x\ u)\ else\ y)\ v) \xrightarrow{\bar{\delta}} ((\lambda x.y)\ v)$.

2.2 Properties of $DLAL_B$

The contraction rule (Cntr) is used only on variables on the l.h.s. of the semi-colon. It is then straightforward to check the following statements:

Lemma 1 *(Free Variable Lemma)*

- *If* $\Gamma;\Delta \vdash t : A$ *then* $FV(t) \subset dom(\Gamma) \cup dom(\Delta)$
- *If* $\Gamma;\Delta \vdash t : A$, $\Delta' \subset \Delta$, $\Gamma' \subset \Gamma$ *and* $FV(t) \subset dom(\Gamma') \cup dom(\Delta')$ *then* $\Gamma';\Delta' \vdash t : A$
- *If* $\Gamma;\Delta \vdash t : A$ *and* $x \in \Delta$ *then we have* $no(x,t) \leqslant 1$

We can make the following remarks on $DLAL_B$ rules:

- Initially the variables are linear (rule (Id)); to convert a linear variable into a non-linear one we can use the (§ i) rule. Note that it adds a § to the type of the result and that the variables that remain linear get a § type too.
- the ($\multimap$ i) (resp. ($\Rightarrow$ i)) rule corresponds to abstraction on a linear variable (resp. non-linear variable);
- observe ($\Rightarrow$ e): a term of type $A \Rightarrow B$ can only be applied to a term u with at most one occurrence of free variable.

Theorem 1 *(Subject Reduction)*
Let $\xrightarrow{\delta\beta} = (\xrightarrow{\beta} \cup \xrightarrow{\delta})$
If $\Gamma;\Delta \vdash t : A$ *is derivable and* $t \xrightarrow{\delta\beta} v$, *then* $\Gamma;\Delta \vdash v : A$.
If $\Gamma;\Delta \vdash t : A$ *is derivable and* $t \xrightarrow{\bar{\delta}\beta} v$, *then* $\Gamma;\Delta \vdash v : A$ *or* $\Gamma;\Delta \vdash v : \S^n\mathbf{Bool}$.

Proof.
Almost the same as in [2].

In order to prove the strong normalisation of the terms well-typed in $DLAL_B$, we will prove that such terms can be translated into terms of System F (which has the property of strong normalisation).

Definition 6 *The translation ()* of a* $DLAL_B$ *type in a type of System F is inductively defined on the structure of the types as follows:* $(\alpha)^* = \alpha$, $(A \multimap B)^* = (A)^* \to (B)^*$, $(A \Rightarrow B)^* = (A)^* \to (B)^*$, $(\S A)^* = (A)^*$, $(\forall \alpha.A)^* = \forall \alpha.(A)^*$, $(Bool)^* = \forall \alpha.\alpha \to \alpha \to \alpha$.

Definition 7 *The translation ()* of a term of* Λ_B *in a term of* Λ *is inductively defined on the structure of the terms as follows:* $(x)^* = x$, $(F)^* = \lambda x.\lambda y.y$, $(T)^* = \lambda x.\lambda y.x$, $(\lambda x.t)^* = \lambda x.(t)^*$, $(t\ u)^* = (t)^*\ (u)^*$, $(if\ t\ then\ u\ else\ v)^* = (t)^*\ (u)^*\ (v)^*$.

Lemma 2 *If* $\Gamma;\Delta \vdash t : A$ *then* $(\Gamma)^*, (\Delta)^* \vdash_F (t)^* : (A)^*$.

Proof. By induction on the structure of the type derivation of t.

Lemma 3 *Let* t *and* t' *be two terms of* Λ_B *such that* $\Gamma;\Delta \vdash t : A$ *and* $t \xrightarrow{\delta\beta} t'$ *then:* $(t)^* \xrightarrow{\beta} (t')^*$, $(\Gamma)^*, (\Delta)^* \vdash_F (t)^* : (A)^*$ *and* $(\Gamma)^*, (\Delta)^* \vdash_F (t')^* : (A)^*$.

Proof. By the definition of the translation of the terms, Lemma 2 and Theorem 1.

Theorem 2 *(Strong Normalisation)*
Let t *be a term of* Λ_B*, if* $\Gamma;\Delta \vdash t : A$ *then* t *is strongly normalizable.*

Proof. By Lemma 3 and the property of strong normalization of terms typeable in System F.

Theorem 3 *(Confluence)*
The $\delta\beta$*-reduction is confluent on the terms of* Λ_B *typeable in* $DLAL_B$.

Proof. By Theorem 2 and the local confluence of the $\delta\beta$-reduction on Λ_B.

Theorem 4 *(Normal Form)*
Let t *be a term of* Λ_B*, if* $\Gamma;\Delta \vdash t : A$ *then* t *has a unique normal form (denoted* $Norm(t)$*).*

Proof. By Theorems 2 and 3.

Lemma 4 *If* $;\vdash t : \S^n\mathbf{Bool}$ *then:*

1. t *is not an abstraction*
2. *if* t *is normal for the* $\beta\delta$*-reduction then* $t = T$ *or* $t = F$.

Proof.

1. By induction on the structure of derivations.

2. By induction on the structure of terms and (i).

2.3 Stratified terms

We have to describe the size of a term in detail in order to better control it during β- and δ-reduction.

Definition 8 *A stratified term is a term with each abstraction symbol* λ *annotated by a natural number* k *(called its depth) and also possibly by symbol !, and with applications possibly annotated by !.*

Thus an abstraction looks like $\lambda^k x.t$ or $\lambda^{k!}x.t$ and an application like $t\ u$ or $t\ !\ u$. When t is a stratified term, $t[+1]$ denotes t with the depths of all abstraction subterms increased by 1. The type assignment rules for stratified terms are obtained by modifying some of the rules of $DLAL_B$ as follows:

$$\frac{\Gamma;\Delta, x : A \vdash t : B}{\Gamma;\Delta \vdash \lambda^0 x.t : A \multimap B}\ (\multimap \text{i}) \qquad \frac{\Gamma, x : A;\Delta \vdash t : B}{\Gamma;\Delta \vdash \lambda^{0!} x.t : A \Rightarrow B}\ (\Rightarrow \text{i})$$

$$\frac{\Gamma;\Delta \vdash t : A \Rightarrow B \quad ;z : C \vdash u : A}{\Gamma, z : C;\Delta \vdash t\ !\ u[+1] : B}\ (\Rightarrow \text{e}) \qquad \frac{;\Gamma,\Delta \vdash t : A}{\Gamma;\S\Delta \vdash t[+1] : \S A}\ (\S \text{ i})$$

The depth of a term is the maximal depth of all the abstractions it contains.

Lemma 5 *Given a* $DLAL_B$ *derivation* $\mathscr{D}$ *of* $\Gamma;\Delta \vdash t : A$ *of depth* d*,* t *can be decorated as a stratified term* t' *of depth* d *such that* $\Gamma;\Delta \vdash t' : A$.

Proof. By induction on the structure of the derivation $\mathscr{D}$.

We can see that $\forall$§-Normalisation Lemma, Abstraction Property Lemma, Paragraph Property Lemma and Subject Reduction Theorem hold for stratified terms as well (as in [2]).

Definition 9 *The number of occurences of symbols λ at depth k in a stratified term is inductively defined on the structure of the terms as follows: $no(k,x)=0$, $no(k,F)=0$, $no(k,T)=0$, $no(k,\lambda^k x.t)=no(k,t)+1$, $no(k,\lambda^p x.t)=no(k,t)$, $no(k,t\ u)=no(k,t)+no(k,u)$, $no(k,if\ t_0\ then\ t_1\ else\ t_2)=\max_i no(k,t_i)$. The definition of the number of occurences of if in a term t, $no(if,t)$, is similar.*

Lemma 6 *Let t be a stratified term such that $\Gamma;\Delta\vdash t:A$ is derivable. If $(v\ !\ u)$ is a subterm of t then:*

- $(FV(u)=\emptyset)$
  or
  $(FV(u)=\{x\}$ *and* $(x\in Dom(\Gamma)$ *or* x *is bound in* t *by a* λ *annotated by* $!)$ *and*
  $no(x,u)=1)$
- *if* $v=\lambda^{k!}x.r$ *then* $\forall p\leq k,\ no(p,u)=0$

Proof. By induction on the structure of the derivation and Lemma 1.

We can now define, with the notations on a stratified term, a vector of integers which characterizes the size of the term.

Definition 10 *Let t be a stratified term,*
we define $vect_d(t)=(no(0,t),\ ...,\ no(d,t),\ no(if,t))$.

Definition 11 *Let a and b be two vectors of $\mathbb{Z}^p$, we define:*

- *$a\leq b$ if and only if $\forall k\leq p,\ a_k\leq b_k$;*
- *$a<b$ if and only if $a\leq b$ and $a\neq b$.*

Lemma 7 *If t and u are two stratified terms such that*
$r=no(x,t)$, $a=vect_d(t)$ and $b=vect_d(u)$, then
$vect_d(t[u/x])\leq a+r*b=(a_0+r*b_0,\ ...,\ a_{d+1}+r*b_{d+1})$.

Proof. By induction on the structure of the term t.

3 APTIME Soundness

Usually, a complexity bound for *LLL* and related systems like *DLAL* is obtained from a specific reduction strategy: the level by level strategy. Such strategy consists to reduce first redexes at level 0 then redexes at level 1 and so on. However, it is not possible to apply such strategy in the λ-calculus with the conditional constructor without breaking the polynomial bound. This is why like Gaboardi and al. we consider a λ-calculus machine to reduce the terms. A delicate point however is that previous work on *LLL* and *DLAL* does not provide complexity bounds on λ-calculus machines. Thus, we need to introduce a suitable measure in order to prove this complexity bound.

3.1 Definitions

Definition 12 *(Programs)*
A program is a term t of Λ_B such that $;\vdash t : \S^n\mathbf{Bool}$.
We define the relation $\leftarrow$ by:

- *If $(Norm(t) = F)$ then $(no \leftarrow t)$;*
- *If $(Norm(t) = T)$ then $(yes \leftarrow t)$.*

Definition 13 *(Contexts)*

- *A context $\mathscr{A}$ is a sequence of variable assignments of the shape $x_i := t_i$ where all variables x_i are distinct. The set of contexts is denoted by C_{tx}.*
- *The cardinality of a context $\mathscr{A}$, denoted by $\#(\mathscr{A})$, is the number of variable assignments in $\mathscr{A}$.*
- *The empty context is denoted by $\emptyset$.*
- *Let $\mathscr{A} = [x_1 := t_1,\ ...,\ x_n := t_n]$ be a context. Then $()^{\mathscr{A}} : \Lambda_B \to \Lambda_B$ is the map that associates the term $t[t_n/x_n]...[t_1/x_1]$ to each term t.*

Definition 14 *(Configurations)*
There is 4 types of configurations:

- *a rejecting configuration: $[\![(Rejecting)]\!]$;*
- *an accepting configuration: $[\![(Accepting)]\!]$;*
- *an existensial configuration: $[\![(\exists)\ \mathscr{A} \mid \{b;\ t\}]\!]$ with $\mathscr{A}$ a context, t a term and $b \in \{yes; no\}$;*
- *a universal configuration: $[\![(\forall)\ \mathscr{A} \mid \{b;\ t\}\ \{b';\ t'\}]\!]$ with $\mathscr{A}$ a context, t and t' two terms and $b, b' \in \{yes; no\}$;*

Definition 15 *The Abstract Alternating Machine $\mathscr{K}_{\mathscr{B}}$ (which is similar to the Krivine machine ([10]) when restricted to the λ-calculus) is a machine that takes as input a program t, starts with the initial configuration $[\![(\exists)\ \emptyset \mid \{yes;\ t\}]\!]$ and reduces t using the two transition functions described in Figure 2. It accepts the program t if its normal form is true and rejects it if its normal form is false (as will be shown below).*

The base cases are obvious. The (β) transition applies when the head of the subject is a β-redex. Then the association between the bound variable and the argument is remembered in context $\mathscr{A}$. The (h) transition replaces the head occurence of the head variable by the term associated with it in the context. The (if) transitions, always followed by the (if') transitions, perform the $\bar{\delta}$ reductions (following the intuition that: $if\ t_0\ then\ t_1\ else\ t_2 = (t_0 \wedge t_1) \vee (\neg t_0 \wedge t_2)$).

Definition 16 *(Computations)*
The computation of the Abstract alternating machine $\mathscr{K}_{\mathscr{B}}$ is the tree obtained by applying the rules given in figure 2 starting from the initial configuration. The definition of a configuration accepted by $\mathscr{K}_{\mathscr{B}}$ and of a computation accepted by $\mathscr{K}_{\mathscr{B}}$ is the same as those of the Alternating Turing Machine.

From here until the end of the subsection 3.3, we will fix a program M. Note that:

- $m = |M|$ (with $|M|$ the size of M);
- $\mathscr{D}$ is a derivation of $;\vdash M : \S^n\mathbf{Bool}$;
- d is the depth of $\mathscr{D}$;

$$\begin{array}{rcl}
[\![(\exists)\ \mathscr{A} \mid \{b;\ \lambda x.N\ N_1\ ...\ N_p\}]\!] & \xrightarrow[(\beta)]{1/2} & [\![(\exists)\ \mathscr{A}@(x' := N_1) \mid \{b;\ N[x'/x]\ N_2\ ...\ N_p\}]\!](*) \\
[\![(\exists)\ \mathscr{A}_1@(x := N)@\mathscr{A}_2 \mid \{b;\ x\ N_1\ ...\ N_p\}]\!] & \xrightarrow[(h)]{1/2} & [\![(\exists)\ \mathscr{A}_1@(x := N)@\mathscr{A}_2 \mid \{b;\ N\ N_1\ ...\ N_p\}]\!] \\
[\![(\exists)\ \mathscr{A} \mid \{b;\ (if\ M_0\ then\ M_1\ else\ M_2)\ N_1\ ...\ N_p\}]\!] & \xrightarrow[(if)]{1} & [\![(\forall)\ \mathscr{A} \mid \{yes;\ M_0\}\ \{b;\ M_1\ N_1\ ...\ N_p\}]\!] \\
[\![(\exists)\ \mathscr{A} \mid \{b;\ (if\ M_0\ then\ M_1\ else\ M_2)\ N_1\ ...\ N_p\}]\!] & \xrightarrow[(if)]{2} & [\![(\forall)\ \mathscr{A} \mid \{no;\ M_0\}\ \{b;\ M_2\ N_1\ ...\ N_p\}]\!] \\
[\![(\forall)\ \mathscr{A} \mid \{a;\ M_0\}\ \{b;\ N\}]\!] & \xrightarrow[(if')]{1} & [\![(\exists)\ \mathscr{A} \mid \{a;\ M_0\}]\!] \\
[\![(\forall)\ \mathscr{A} \mid \{a;\ M_0\}\ \{b;\ N\}]\!] & \xrightarrow[(if')]{2} & [\![(\exists)\ \mathscr{A} \mid \{b;\ N\}]\!]
\end{array}$$

base	$[\![(\exists)\ \mathscr{A} \mid \{yes;\ T\}]\!] \xrightarrow{1/2} [\![(Accepting)]\!]$	$[\![(\exists)\ \mathscr{A} \mid \{no;\ F\}]\!] \xrightarrow{1/2} [\![(Accepting)]\!]$
cases	$[\![(\exists)\ \mathscr{A} \mid \{no;\ T\}]\!] \xrightarrow{1/2} [\![(Rejecting)]\!]$	$[\![(\exists)\ \mathscr{A} \mid \{yes;\ F\}]\!] \xrightarrow{1/2} [\![(Rejecting)]\!]$

(*) x' is a fresh variable. $1/2$ means 1 or 2.

Figure 2: The Rules of the Abstract Alternating Machine $\mathscr{K}_{\mathscr{B}}$

- M' is the stratified term of depth d associated with the term M;
- $r = \max_x no(x,M)$ (with $r < m$ by definition).

Definition 17 *Let* $t_k,\ u_k,\ v_k : \mathbb{Z}^{d+2} \to \mathbb{Z}^{d+2}$ *such that:*

- $t_k(a) = (a_0,\ ...,\ a_{k-1},\ a_k - 1,\ a_{k+1} + r(b_{k+1} + m),\ ...,\ a_{d+1} + r(b_{d+1} + m))$;
- $u_k(a) = (a_0,\ ...,\ a_{k-1},\ a_k - 1,\ a_{k+1},\ ...,\ a_{d+1})$;
- $v_k(b) = (b_0,\ ...,\ b_k,\ b_{k+1} + m,\ ...,\ b_{d+1} + m)$.

We want to establish a complexity bound on the machine. For that, we define a measure on the vectors characterizing the size of terms such that this measure will decrease with β- and δ-reduction.

Definition 18 *Let* $measure_{(i)} : \mathbb{Z}^{i+2} \times \mathbb{Z}^{i+2} \to \mathbb{Z}$ *such that:*

- $measure_{(-1)}(a_0,\ b_0) = a_0$
- $measure_{(i+1)}((a_0,\ ...,\ a_{i+2}),\ (b_0,\ ...,\ b_{i+2})) =$
  $measure_{(i)}((a_1 + (r+1)(b_1 + a_0 * m)a_0,\ ...,\ a_{i+2} + (r+1)(b_{i+2} + a_0 * m)a_0),$
  $(b_1 + a_0 * m,\ ...,\ b_{i+2} + a_0 * m))$.

Lemma 8 $\forall k \geq -1,\ \forall a,b,a',b' \in \mathbb{N}^{k+2}$,
if $a' \leq a$ *and* $b' \leq b$ *then* $measure_{(k)}(a',\ b') \leq measure_{(k)}(a,\ b)$;
if $a' \leq a,\ b' \leq b$ *and* $a' \neq a$ *then* $measure_{(k)}(a',\ b') < measure_{(k)}(a,\ b)$.

Proof. By definition of *measure*.

Lemma 9 *Let* $a,b \in \mathbb{N}^{d+2}$.

- $\forall k \in [\![0;\ d]\!],\ if\ t_k(a), v_k(b) \in \mathbb{N}^{d+2}\ then\ 0 \leq measure_{(d)}(t_k(a),\ v_k(b)) < measure_{(d)}(a,\ b)$;
- $\forall k \in [\![0;\ d+1]\!],\ if\ u_k(a) \in \mathbb{N}^{d+2}\ then\ 0 \leq measure_{(d)}(u_k(a),\ b) < measure_{(d)}(a,\ b)$.

Proof. By Lemma 8 and definitions of *measure*, t, *u and* v.

$$\frac{[\![(i+1)\ (\mathscr{A}@(x_i^! := N_1) \mid N[x_i^!/x]\ N_2\ ...\ N_p)\ (t_k(a),\ v_k(b))]\!]}{[\![(i)\ (\mathscr{A} \mid (\lambda^{k!}x.N\ !\ N_1)\ ...\ N_p)\ (a,\ b)]\!]}\ (\beta!)$$

$$\frac{[\![(i+1)\ (\mathscr{A}@(x_i := N_1) \mid N[x_i/x]\ N_2\ ...\ N_p)\ (u_k(a),\ b)]\!]}{[\![(i)\ (\mathscr{A} \mid \lambda^k x.N\ N_1\ ...\ N_p)\ (a,\ b)]\!]}\ (\beta)$$

$$\frac{[\![(i)\ (\mathscr{A}_1@(x := N)@\mathscr{A}_2 \mid N\ N_1\ ...\ N_p)\ (a,\ b)]\!]}{[\![(i)\ (\mathscr{A}_1@(x := N)@\mathscr{A}_2 \mid x\ N_1\ ...\ N_p)\ (a,\ b)]\!]}\ (\text{h})$$

$$\frac{[\![(i)\ (\mathscr{A} \mid M_0)\ (u_{d+1}(a),\ b)]\!] \quad [\![(i)\ (\mathscr{A} \mid M_1\ N_1\ ...\ N_p)\ (u_{d+1}(a),\ b)]\!] \quad [\![(i)\ (\mathscr{A} \mid M_2\ N_1\ ...\ N_p)\ (u_{d+1}(a),\ b)]\!]}{[\![(i)\ (\mathscr{A} \mid (if\ M_0\ then\ M_1\ else\ M_2)\ N_1\ ...\ N_p)\ (a,\ b)]\!]}\ (\text{if})$$

$$\frac{[\![(0)\ (\emptyset \mid M')\ ((m,\ ...,\ m),\ (0,\ ...,\ 0))]\!]}{}\ (\text{root})$$

Figure 3: The Rules of the Transformation Tree of M: $\mathscr{T}(M)$

Definition 19 *The Transformation Tree of M, $\mathscr{T}(M)$, describes the computation tree of $\mathscr{K}_{\mathscr{B}}$ on the input M and contains nodes which are elements of the set $\mathbb{N} \times (Ctx \times \Lambda_B) \times (\mathbb{N}^{d+2} \times \mathbb{N}^{d+2})$. This tree is inductively defined by the rules given in Figure 3. Note that the terms in this tree are decorated (these terms are stratified terms).*

$\mathscr{T}(M)$ will be used to bound the time of computation of $\mathscr{K}_{\mathscr{B}}$ on M.

3.2 APTIME soundness of $\mathscr{K}_{\mathscr{B}}$

Lemma 10 *Let $[\![(j),\ (\mathscr{A}|t),\ (a,\ b)]\!]$ be a node of $\mathscr{T}(M)$.*

1. *for each $(x_i^! := t_i) \in \mathscr{A}$, $vect_d((t_i)^{\mathscr{A}}) \leq b$.*
2. *$vect_d((t)^{\mathscr{A}}) \leq a$.*

Proof.

1. By induction on the structure of the tree using Lemmas 6 and 7.
2. By induction on the structure of the tree using (1) and Lemma 7
   (given that all the elements of the canonical composition of t and all the terms of $\mathscr{A}$ are subterms of M).

Lemma 11 *Let $n = [\![(j),\ (\mathscr{A}|t),\ (a,\ b)]\!]$ and $n' = [\![(j'),\ (\mathscr{A}'|t'),\ (a',\ b')]\!]$ be two nodes of $\mathscr{T}(M)$. If n' is a son of n linked by a rule $(\beta!)$, (β) or (if), then $0 \leq measure_{(d)}(a',\ b') < measure_{(d)}(a,\ b)$.*

Proof. By Lemma 10 we have $a, b, a', b' \in \mathbb{N}^{d+2}$, thus by Lemma 9 we have $0 \leq measure_{(d)}(a',\ b') < measure_{(d)}(a,\ b)$.

Definition 20 *Let n be a node of $\mathscr{T}(M)$.*

- *$\#_{\beta!}(n)$ denotes the number of applications of the $(\beta!)$ rule in the path between the root of $\mathscr{T}(M)$ and n.*
- *$\#_{\beta}(n)$ denotes the number of applications of the (β) rule in the path between the root of $\mathscr{T}(M)$ and n.*

- $\#_h(n)$ *denotes the number of applications of the* (h) *rule in the path between the root of* $\mathscr{T}(M)$ *and* n.
- $\#_{if}(n)$ *denotes the number of applications of the* (if) *rule in the path between the root of* $\mathscr{T}(M)$ *and* n.

Lemma 12 *Let* n *be a node of* $\mathscr{T}(M)$.

1. $\#_{\beta!}(n)+\#_\beta(n)+\#_{if}(n) \leq measure_{(d)}((m,\ ...,\ m),\ (0,\ ...,\ 0))$
2. $\#_h(n) \leq (measure_{(d)}((m,\ ...,\ m),\ (0,\ ...,\ 0)))^2$

Proof.

1. By Lemma 11.
2. Let n' and n'' two nodes of $\mathscr{T}(M)$ such that there is a path of applications of the (h) rule from n' to n''.
   $\#_h(n'')-\#_h(n') \leq \#(\mathscr{A}_{n'}) = \#_{\beta!}(n')+\#_\beta(n')$.
   Thus $\#_h(n) \leq (\#_{\beta!}(n)+\#_\beta(n)) * (\#_{\beta!}(n)+\#_\beta(n)+\#_{if}(n))$.

Lemma 13 $\forall i \in [\![1;\ d+1]\!]$,
$measure_{(d)}((m,...,m),\ (0,...,0)) \leq measure_{(d-i)}((m^{3^{i+1}},...,m^{3^{i+1}}),\ (m^{3^i+1},...,m^{3^i+1}))$.

Proof. By induction on i using Lemma 8.

Theorem 5 *The machine* $\mathscr{K}_{\mathscr{B}}$ *on the input* M *is computing in a time bounded by* $m^{3^{d+3}}$.

Proof. By Lemma 13, $measure_{(d)}((m,\ ...,\ m),\ (0,\ ...,\ 0)) \leq m^{3^{d+2}}$.
Furthermore by Lemma 12, let n be a node of $\mathscr{T}(M)$,
let $k = measure_{(d)}((m,\ ...,\ m),\ (0,\ ...,\ 0))$ and let $p_n = \#_{\beta!}(n)+\#_\beta(n)+\#_{if}(n)+\#_h(n)$,
$p_n \leq k+k^2$.
Thus $Time(\mathscr{K}_{\mathscr{B}}(M)) \leq 2 * Depth(\mathscr{T}(M))+1 = 2*(\max_n p_n)+1 \leq 2*(m^{3^{d+2}}+m^{2*3^{d+2}})+1$.

Note that the measure we have defined can be applied in the restricted case of *DLAL* programs: in this case the machine is deterministic and our measure gives a new proof that *DLAL* terms of boolean type can be evaluated in polynomial time where the degree of the polynomial depends on the depth of the term.

3.3 Correctness of $\mathscr{K}_{\mathscr{B}}$

Now, we need to prove that the alternating λ-calculus machine computes the right value.

Lemma 14 *If* $[\![(\exists)\ \mathscr{A} \mid \{b;\ t\}]\!]$ *(resp.* $[\![(\forall)\ \mathscr{A} \mid \{b;\ t\}\ \{b';\ t'\}]\!]$*) is a configuration of the computation of* $\mathscr{K}_{\mathscr{B}}$ *on* M *then* $;\vdash (t)^{\mathscr{A}} : \S^n\mathbf{Bool}$ *(resp.* $;\vdash (t)^{\mathscr{A}} : \S^n\mathbf{Bool}$ *and* $;\vdash (t')^{\mathscr{A}} : \S^{n'}\mathbf{Bool}$*).*

Proof. By induction on the structure of the tree using Theorem 1.

Lemma 15 *If* $c = [\![(\exists)\ \mathscr{A} \mid \{b;\ t\}]\!]$ *(resp.* $[\![(\forall)\ \mathscr{A} \mid \{b;\ t\}\ \{b';\ t'\}]\!]$*) is a configuration of the computation of* $\mathscr{K}_{\mathscr{B}}$ *on* M *then* $\mathscr{K}_{\mathscr{B}}$ *is accepting* c *if and only if* $b \leftarrow (t)^{\mathscr{A}}$ *(resp.* $b \leftarrow (t)^{\mathscr{A}}$ *and* $b' \leftarrow (t')^{\mathscr{A}}$*).*

Proof. By induction on the tree, starting from the leafs and using Lemmas 14 and 4.

Theorem 6 *The machine* $\mathscr{K}_{\mathscr{B}}$ *is accepting* M *if and only if* $Norm(M) = T$.

$$\frac{\Gamma_1;\Delta_1 \vdash t_1 : A_1 \quad \Gamma_2;\Delta_2 \vdash t_2 : A_2}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash t_1 \otimes t_2 : A_1 \otimes A_2} \ (\otimes \text{ i})$$

$$\frac{\Gamma_1;\Delta_1 \vdash u : A_1 \otimes A_2 \quad \Gamma_2;x_1 : A_1, x_2 : A_2, \Delta_2 \vdash t : B}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \mathsf{let}\ u\ \mathsf{be}\ x_1 \otimes x_2\ \mathsf{in}\ t : B} \ (\otimes \text{ e})$$

Figure 4: Derived rules

Proof. By Lemma 15.

If t is a closed term of type $\mathbf{W} \Rightarrow \S^n\mathbf{Bool}$, we define $\mathscr{L}(t)$ as the set of words accepted by t. Finally, we obtain the desired result:

Theorem 7 (APTIME soundness of $DLAL_B$) *Let t be a term of Λ_B such that $;\vdash t : \mathbf{W} \Rightarrow \S^n\mathbf{Bool}$ or $;\vdash t : \mathbf{W} \multimap \S^n\mathbf{Bool}$ has a derivation of depth d.*
Let $\mathscr{M}$ be the Alternating Turing Machine which, on the input i represented by the λ-term w, simulates the machine $\mathscr{K}_{\mathscr{B}}$ on the input $(t\ w)$.
Then $\mathscr{M}$ decides the language represented by t and $\mathscr{M}$ is computing in time $O(m^{3^{d+4}})$.
Thus $\mathscr{L}(t) \in APTIME$.

Proof. By Theorems 6 and 5.

4 APTIME Completeness

This section presents the second part of the proof that $DLAL_B$ characterizes the predicates of PSPACE and is simply using classical ideas of the literature (see [3] and [7]).

We have the following data types for unary integers and binary words in $DLAL_B$:

$$\begin{aligned} \mathbf{N} &= \forall\alpha.(\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha), \\ \mathbf{W} &= \forall\alpha.(\alpha \multimap \alpha) \Rightarrow (\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha). \end{aligned}$$

The inhabitants of types $\mathbf{N}$ and $\mathbf{W}$ are the familiar Church codings of integers and words:

$$\begin{aligned} \underline{n} &= \lambda f.\lambda x.\underbrace{f(f\ldots(f}_{n}x)\ldots), \\ \underline{w} &= \lambda f_0.\lambda f_1.\lambda x.f_{i_1}(f_{i_2}\ldots(f_{i_n}x)\ldots), \end{aligned}$$

with $i \in \{0,1\}$, $n \in N$ and $w = i_1 i_2 \cdots i_n \in \{0,1\}^*$.

It can be useful in practice to use a type $A \otimes B$. It can be defined anyway, thanks to full weakening:

$$A \otimes B = \forall\alpha.((A \multimap B \multimap \alpha) \multimap \alpha).$$

We use as syntactic sugar the following new constructions on terms with the typing rules of Figure 4:

$$\begin{aligned} t_1 \otimes t_2 &= \lambda x.xt_1t_2, \\ \mathsf{let}\ u\ \mathsf{be}\ x_1 \otimes x_2\ \mathsf{in}\ t &= u(\lambda x_1.\lambda x_2.t). \end{aligned}$$

Theorem 8 (APTIME completeness of $DLAL_B$) *If a function $f : \{0,1\}^\star \to \{0,1\}$ is computable in time n^{2^d} by a one-tape alternating Turing machine for some d, then there exists a term M of Λ_B such that $;\vdash M : \mathbf{W} \Rightarrow \S^{2d+2}\mathbf{Bool}$ and M represents f.*

Proof (Sketch). Let $\mathscr{M}$ be an ATM with 2 symbols, 1 tape, k classical states and four characteristic states. The four characteristic states: Accepting, Rejecting, Universal and Existential, are represented respectively by $A = F \otimes T$, $R = F \otimes F$, $\wedge = T \otimes F$ and $\vee = T \otimes T$ of type $\mathbf{Bool}^2$.

Following the idea of [1], let **Conf** be the $DLAL_B$-type

$$\forall\alpha.(\alpha \multimap \alpha) \Rightarrow (\alpha \multimap \alpha) \Rightarrow \S((\alpha \multimap \alpha)^2 \otimes (\mathbf{Bool}^k \otimes \mathbf{Bool}^2)),$$

which serves as a type for the configurations of the ATM.

We will proceed in the same way as Gaboardi et al.:

- show that all polynomials can be represented in the system;
- define a function Step which answers recursively if a configuration will be accepted or not by the ATM, it will be given the type:
  $(\mathbf{Conf} \multimap \mathbf{Bool}^2) \multimap (\mathbf{Conf} \multimap \mathbf{Bool}^2)$;
- define a term which decides if a given configuration is accepted, by iterating Step a polynomial number of times.

We have the following $DLAL_B$-terms:

- trans_1 (*resp.* trans_2) : **Conf** $\multimap$ **Conf** for one-step of the first (resp. the second) function of transition of the ATM (similar to trans in [2]);
- Kind : **Conf** $\multimap \mathbf{Bool}^2$ for the projection from a configuration to its characteristic state;
- $\mathsf{P} : \mathbf{N} \multimap \S^{2d}\mathbf{N}$ for the polynomial $n \mapsto n^{2^d}$ (same as P in [2]).

The term Step (of type $(\mathbf{Conf} \multimap \mathbf{Bool}^2) \multimap (\mathbf{Conf} \multimap \mathbf{Bool}^2)$) is defined in a way analogous to Step in [7]:

- $\mathsf{Term}_3 = if\ \pi_2\ (h\ (\mathsf{trans}_1\ c))\ then\ F \otimes (\pi_2\ (h\ (\mathsf{trans}_2\ c)))\ else\ R$;
- $\mathsf{Term}_2 = if\ \pi_2\ (h\ (\mathsf{trans}_1\ c))\ then\ A\ else\ F \otimes (\pi_2\ (h\ (\mathsf{trans}_2\ c)))$;
- $\mathsf{Term}_1 = if\ \pi_2\ (\mathsf{Kind}\ c)\ then\ \mathsf{Term}_2\ else\ \mathsf{Term}_3$;
- $\mathsf{Step} = \lambda h.\lambda c.if\ \pi_1\ (\mathsf{Kind}\ c)\ then\ \mathsf{Term}_1\ else\ \mathsf{Kind}\ c$.

Step term operation:

- If Step receives as argument a configuration c and a function of characterization h of type **Conf** $\multimap$ $\mathbf{Bool}^2$ such that $h\ (\mathsf{trans}_1\ c)$ (resp. $h\ (\mathsf{trans}_2\ c)$ returns A if $\mathsf{trans}_1\ c$ is accepted by the ATM and R if it is rejected (resp. A if $\mathsf{trans}_2\ c$ is accepted by the ATM and R if it is rejected) then $\mathsf{Step}\ h\ c$ returns A if c is accepted by the ATM and R if it is rejected;
- Term_1 represents the case where the characteristic state of c is neither Accepting nor Rejecting (in which cases it is sufficient to return $\mathsf{Kind}\ c$);
- Term_2 (resp. Term_3) represents the case where the characteristic state of c is Existential (resp. Universal).

We also have the following $DLAL_B$-terms:

- init : **W** $\multimap$ **Conf** for initialization (similar to init in [2]);

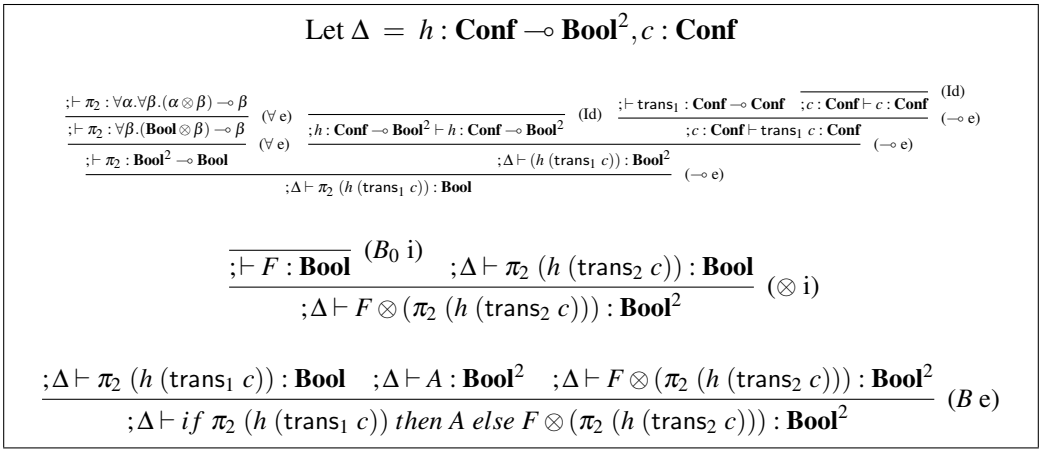

Figure 5: Type derivation for the term Term_2

- $\mathsf{length} : \mathbf{W} \multimap \mathbf{N}$ for the length map (similar to length in [2]);
- $\mathsf{coer} : \mathbf{W} \multimap \S^{2d}\mathbf{W}$ for an identity function (usefull for the typing and similar to coer in [2]).

Finally we have $\mathsf{M} : \mathbf{W} \multimap \S^{2d+2}\mathbf{Bool}$ which is the term representing the ATM $\mathscr{M}$: $\mathsf{M} = \lambda w.(\pi_2\ (\mathsf{P}\ (\mathsf{length}\ w)\ \mathsf{Step}\ \mathsf{Kind}$ M term operation:

- $\mathsf{init}\ (\mathsf{coer}\ w))$ is a term which represents the initial configuration of the ATM.
- Step calls itself recursively n^{2^d} times (with n, represented by length w, the length of the word w) -thanks to the term $\mathsf{P}\ (\mathsf{length}\ w)$- so that it calls Kind only on configurations which have a characteristic state Accepting or Rejecting. Thus the term $\mathsf{P}\ (\mathsf{length}\ w)\ \mathsf{Step}\ \mathsf{Kind}\ (\mathsf{init}\ (\mathsf{coer}\ w))$ returns A if w is accepted by the ATM and R if it is rejected.
- Therefore $\pi_2\ (\mathsf{P}\ (\mathsf{length}\ w)\ \mathsf{Step}\ \mathsf{Kind}\ (\mathsf{init}\ (\mathsf{coer}\ w)))$ returns T (true) if w is accepted by the ATM and F (false) if it is rejected. Thus M represents $\mathscr{M}$.

5 Conclusion and perspectives

We have presented a polymorphic type system for lambda calculus with booleans which guarantees that all well-typed terms are representing APTIME predicates and that all predicates of APTIME are represented by well-typed terms. Thus this system is characterizing PSPACE (given that PSPACE = APTIME).
Otherwise, if we were to consider terms of type $\mathbf{W} \Rightarrow \S^n\mathbf{W}$ instead of terms of type $\mathbf{W} \Rightarrow \S^n\mathbf{Bool}$ we believe that we would obtain a characterization of FPSPACE without changing the type assignment system and with the same data type in input and output (which is a property not shared by STA_B).
Now, it would be interesting to see if system $DLAL_B$ could be modified in order to characterize the polynomial hierarchy (PH). We think that such study would be facilitated by the use of APTIME Abstract Machine in the Soundness part of the proof of this paper. Thus this proof could be reused to prove the PH soundness of the modified system of $DLAL_B$.

References

[1] A. Asperti & L. Roversi (2002): *Intuitionistic light affine logic*. ACM Transactions on Computational Logic 3(1), pp. 1–39, doi:10.1145/504077.504081.

[2] P. Baillot & K. Terui (2009): *Light types for polynomial time computation in lambda-calculus*. Information and Computation 207(1), pp. 41–62, doi:10.1016/j.ic.2008.08.005.

[3] Patrick Baillot & Kazushige Terui (2004): *Light types for polynomial time computation in lambda-calculus*. In: Proceedings of LICS'04, pp. 266–275, doi:10.1109/LICS.2004.1319621.

[4] Stephen Bellantoni & Stephen A. Cook (1992): *A New Recursion-Theoretic Characterization of the Polytime Functions*. In: Proceedings of STOC'92, ACM, pp. 283–293.

[5] A.K. Chandra, D.C. Kozen & L.J. Stockmeyer (1981): *Alternation*. Journal of the ACM 28(1), pp. 114–133.

[6] M. Gaboardi & S. Ronchi Della Ronca (2007): *A soft type assignment system for λ-calculus*. In: Proceedings of CSL'07, Springer, doi:10.1007/978-3-540-74915-8_21.

[7] M. Gaboardi, S. Ronchi Della Ronca & J-Y Marion (2008): *A Logical Account of PSPACE*. In: Proceedings of POPL'08, doi:10.1145/1328438.1328456.

[8] J.-Y. Girard (1998): *Light Linear Logic*. Information and Computation 143, pp. 175–204.

[9] Martin Hofmann (2002): *The strength of non-size increasing computation*. In: Proceedings of POPL'02, ACM, pp. 260–269, doi:10.1145/503272.503297.

[10] J-L Krivine (2007): *A call-by-name lambda calculus machine*. Higher Order and Symbolic Computation .

[11] Y. Lafont (2004): *Soft Linear Logic and polynomial time*. Theoretical Computer Science 318(1–2), pp. 163–180, doi:10.1016/j.tcs.2003.10.018.

[12] Jean-Yves Marion & Daniel Leivant (1995): *Ramified Recurrence and Computational Complexity II: Substitution and Poly-Space*. In: Proceedings of CSL'94, Springer, pp. 486–500, doi:10.1007/BFb0022277.

[13] Jean-Yves Marion & Jean-Yves Moyen (2000): *Efficient First Order Functional Program Interpreter with Time Bound Certifications*. In: Proceedings of LPAR'00, Springer, pp. 25–42, doi:10.1007/3-540-44404-1_3.